From the Event Log of a Social Simulation to Narrative Discourse: Content Planning in Story Generation

Carlos León and **Samer Hassan** and **Pablo Gervás** [1]

Abstract.

This paper presents a proposal for implementing automated story telling of narrative threads within a multiplayer game based on selection and linearization of game logs. Our initial prototype operates on logs generated artificially by a social simulation built by a multiagent system. This provides a log of events for a large set of characters emulating real life behaviour over a certain period of time, with no need to carry out a real game involving several players over an equivalent time. The proposed method addresses tasks of content determination - filtering the non-relevant events out of the total log -, and discourse planning - organizing a possibly large set of parallel threads of events into a linear narrative discourse. Actual sentence planning and realization is not addressed, but rather performed in a crude manner to allow readable presentation of the generated material. Examples of system input and output are presented, and their relative merits are discussed. The final section discusses futures lines of work that may be worth exploring.

1 Introduction

Narrative games used for educational purposes have a great potential for improving the learning experience for students, both in terms of making it more interactive and by providing a strong entertainment component that might act as additional motivation. Part of this potential lies in the fact that there is a story underlying the game. This story is in most cases only implicit, in the sense that it arises as the game goes on. This is what makes it interactive, and it presents advantages from the point of view of entertainment. However, from a pedagogical standpoint, having access to an explicit version of the same story may provide additional advantages. On one hand, it may provide the student with a textual summary of how a particular game or gaming session developed. This may be of use when revising material that has already been covered, or in trying to understand what went wrong. The ability to revise is an important ingredient of the learning experience. If games are to take the role currently played by lectures or laboratory sessions, the explicit narratives of such games might play the part of the notes usually taken by students - as game players are unlikely to take notes as they play. On the other hand, explicit narratives reviewing particular sections of a game may constitute a useful tool in developing functionalities for assisting student/players in succesfully completing the game, maybe by explaining how a particular situation in which they find themselves has come about. It is common for current games to have a set level of difficulty, so that part of their entertainment value lies in the challenge of reaching the level of profficiency required. Players setting off to achieve it from a low level of proficiency may have a hard time at the initial stages, up to the point where many give up before achieving the goal. Providing the system with help facilities based on inserting small narratives explaining particular details required for solving puzzles may be seen as detracting from the challenge the game presents as means of entertainment, but they can be a positive addition from the pedagogical point of view if they ensure that more of the students setting out to solve the game actually reach the final goals. To make the point clear, an example is presented for a particular type of game. Some modern games, like MMORPGs[2], are played by several players over huge maps with many locations and many characters. These games usually have different agents interacting between them, and creating more or less complex relations that could be important for the global story of the gameplay. Non-player characters with coherent storylines, set in motion by the casual presence of one player, may meet other players at a later point. In order to understand their behaviour, this second player may need to know their story. This information is actually available in game logs, and it can be read by game masters, which can then write this data in a human readable form. If the system is to manage this task in an autonomous manner, capabilities for automated story telling must be provided. This paper presents a proposal for implementing such functionality: this text in natural language explaining the most interesting parts of the game can be generated by machines resorting to state of the art natural language generation technologies. The actual sequence of events that have happened is available, stored as a system log or in short-term memory. But telling it in an entertaining way, while at the same time filling in the gaps in the players knowledge of what has happened, is not a trivial task. Research in automated telling of stories attempts to fill this gap. The tasks involved will cover the basic requirements for identifying the most relevant material among a large search space of recorded events, converting a sequence of such events - or various parallel sequences of them - into a story, and presenting this selection to the user, already organised into narrative threads.

In order to avoid the task of collecting real data from massive multiplayer online games, we have based our initial prototype on a social simulation generated by a multiagent system. This provides a log of events for a large set of characters emulating real life behaviour over a certain period of time. The simulation we have used was initially developed for a different purpose in the field of experimental social sciences, and it has been adapted to its current purpose by customising the domain characteristics and the set of possible operations available to the agents to simulate a game-like environment.

We want to simulate a game system with many agents or game characters where the main key is the interaction between them, and

[1] Department of Software Engineering and Artificial Intelligence. Universidad Complutense de Madrid.
cleon@fis.ucm.es, samer@fdi.ucm.es, pgervas@sip.ucm.es

[2] Massive Multiplayer Online Role Playing Games

the result is the emergent behaviour as a social group. This behaviour is a full story along a defined period of time, with interesting episodes, boring ones, communities trying to survive, and individual characters doing incredible things. We propose a multi-agent system with social capabilities, emulating a real fantasy medieval game. We have developed a multi-agent system that simulates a community of non-player characters being born, living and dying, where each agent or character saves its history. When all these histories are generated as data logs, we process them to build a structure where only the important facts are told, and that can be easily translated to natural language, freeing in this way the game masters or system administrators from writing this text themselves.

2 Previous Work

In order to develop this system, we have resorted to previous work in the fields of natural language generation and social simulations using multi-agent systems. A brief outline of the relevant studies is given in this section.

2.1 Automatic story generation

The general process of text generation takes place in several stages, during which the conceptual input is progressively refined by adding information that will shape the final text [9]. During the initial stages the concepts and messages that will appear in the final content are decided (*content determination*) and these messages are organised into a specific order and structure (*discourse planning*), and particular ways of describing each concept where it appears in the discourse plan are selected (*referring expression generation*). This results in a version of the discourse plan where the contents, the structure of the discourse, and the level of detail of each concept are already fixed. The *lexicalization* stage that follows decides which specific words and phrases should be chosen to express the domain concepts and relations which appear in the messages. A final stage of *surface realization* assembles all the relevant pieces into linguistically and typographically correct text. These tasks can be grouped into three separate sets: *content planning*, involving the first two, *sentence planning*, involving the second two, and *surface realization*.

The work presented in this paper is related to the first two tasks: content determination and discourse planning. Content determination is known to be always heavily dependent on the particular domain of operation, and tightly coupled with the particular kind of input being processed. Little generalization is possible for this task. Discourse planning determines the ordering and rhetorical relations of the logical messages, hereafter called facts, that the generated document is intended to convey. Most existing approaches to discourse planning are based on either rhetorical structure theory (RST) [5, 4] or schemata [6].

2.2 Social systems

Social phenomena are extremely complicated and unpredictable, since they involve complex interaction and mutual interdependence networks. Sociologic explanations deal with large complex models, with so many dynamic factors involved, they are not subject to laws, but to trends, which can affect individuals in a probabilistic way.

A social system consists of a collection of individuals that interact among them, evolving autonomously and motivated by their own beliefs and personal goals, and the circumstances of their social environment. Due to the mentioned complexity, techniques are required that consider how global behaviour can be derived from the real subjects' behaviours, which are fundamental in any social system. In particular, there is an interest in observing the emergent behaviour that results from the interactions of individuals as a way to discover and analyse the construction and evolution of social patterns.

A multi-agent system (MAS) consists of a set of autonomous software entities (the agents) that interact among them and with their environment. Autonomy means that agents are active entities that can take their own decisions. The agent paradigm assimilates quite well to the individual in a social system. In fact, there are numerous works in agent theory on organisational issues of MAS. Also, theories from the field of Psychology have been incorporated to design agent behaviour, being the most extended the Believes-Desires-Intentions (BDI) model, in the work of [2].

With this perspective, agent-based simulation tools have been developed in recent years to explore the complexity of social dynamics. In this way agents' reactions can be monitored in an observable environment, defining the lines of system evolution. This provides a platform for empirical studies of social systems. And because of that, the specification of characteristics and behaviour of each agent is critical, so it can manage the dimensions of the studied problem. A screenshot of one of these tools is shown in Figure 1.

In the MAS designed, as explained in [7], the agents have been developed with several main attributes: from simple ones such as sex or age, to complex ones, like for example ideology or educational level. The population in the agents' society (as in real societies) also experiments demographic changes: individuals are subject to some life-cycle patterns: they get married, reproduce and die, going through several stages where they follow some intentional and behavioural patterns.

Moreover, the agents/individuals can build and be part of relational groups with other agents: they can communicate with other close agents, leading to friendship relationships determined by the rate of similarity. Or, on the other hand, they can build family nuclei as children are born close to their parents.

Thanks to the underlying sociological model, the parameters of the social simulation system fit all together logically. In this way, the system may be configured to reflect the parameters (such as average number of children per couple, or mean of male average age of death) from a specific country or even import data from surveys that specify the attributes of the agents, reflecting the behaviour of the given population.

Besides, due to the relative simplicity of the agents, the system can manage hundreds of them, reaching the necessary amount for observing an emergent behaviour that results from the interactions of individuals, leading to the appearance of social patterns than can be studied. And for this study, during and after the execution of the simulation tool several graphs may be plotted that reflect the evolution of the main attributes of the social system.

3 Story Generation

Our approach to story generation is based on three tasks: *content determination*, *discourse planning* and *sentence planning*:

- In *content determination* we choose which data is going to be useful for the final narration. In this stage we suppress irrelevant facts present in the log, obtaining a version where redundant or useless data is removed. We can see this step as a "filter" of the log.
- *Discourse planning* consists on identifying a proper order of presentation of the previous data. We apply a particular technique (we

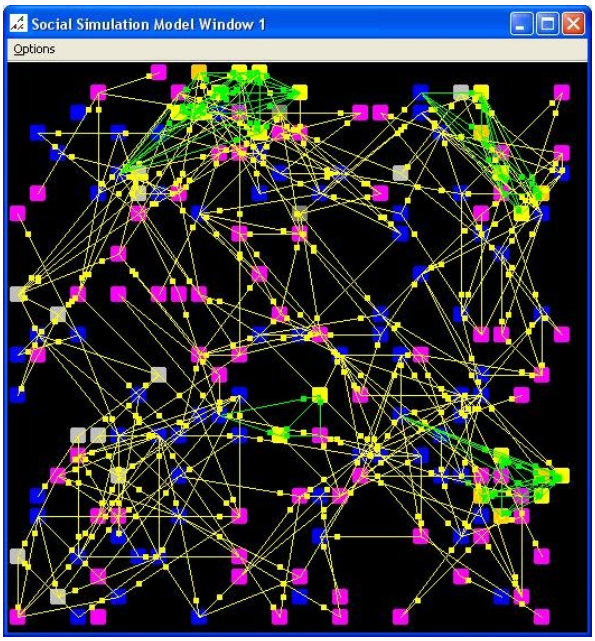

Figure 1. Screenshot of the social simulation

can use several algorithms, later this will be explained), and give the selected data generated in the *content determination* stage a particular order of narration, considered interesting for the readers of the final text.

- Then, we can perform *sentence planning*. This last step is the final process to be done, where the ordered log that represents a story in a structured form is translated to a natural language text.

It is not necessary to run these steps in sequential order. We have decided to join the two first steps into a single one; however, they could be done separated. Next we explain the solutions we have used for this work for each of these previous steps.

3.1 A Manual Story Generation Tool

Before creating a fully automatic system, we want to know which rules we, as humans, apply in story generation. That is the reason why we have created a tool for manual story generation, *Herodotus*. With this tool it is possible, with a simple few mouse clicks, to "draw" a full discourse from the facts and the logs recorded during an execution of a multi-character system.

With *Herodotus* it is possible to perform *content determination*, excluding from the final story those facts that we consider to be boring or not relevant; *discourse planning*, creating the components needed to define a particular narration: relationships between facts (nexus between consecutive facts, like "while", "then" or "before that"), *discourse atoms*, or blocks of facts which are a semantic units (can be seen as paragraphs) and start and end points of the story; and simple *sentence planning*, with template-based solutions for transforming facts into text. This tool can also export a file in each step, in this way, for example, we could do *content determination* and *discourse planning*, export the result, and run a different program to generate natural language text, or an animated summarised reproduction of the gameplay.

To use *Herodotus* one only needs to load an XML file from the multi-agent system or from the log of a real game. Then, the full list of logs for each agent/player becomes visible in the main panel, with their facts, ordered by time. Once loaded, the log can be edited just by dragging with the mouse, drawing lines that represent relationships between the facts.

The facts can also be removed from the list, as well as the full logs, just by selecting them by clicking over them with the mouse, and pressing a button on the toolbar. Also, logs and facts can be added by hand, creating new threads of action and new characters.

Once we have connected the facts in order, and having removed those facts that are not important, it is only needed to group the events in blocks, that will be the *discourse atoms*, as we have explained before.

In Figure 2 we can see a screen capture of *Herodotus* working.

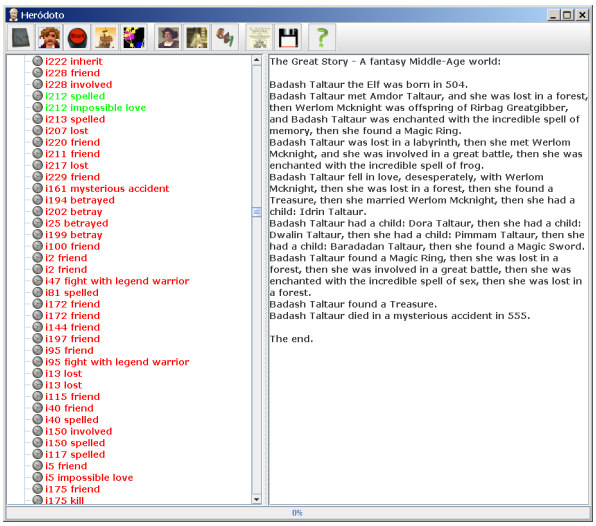

Figure 2. Screenshot of *Herodotus*

3.2 Adapting the MAS for Story Generation

The ideas expressed above concerning social simulations using multiagent systems are the core of action from which we have built the whole narrative system. Several changes to the original MAS have to be made in the perspective of execution to be able to generate full logs of action which will be the basis for the texts describing the storyline. It is necessary to shift the point of view from data acquisition to log generation. These logs must save the data in such a way that story generation can be carried out as easily as possible. We do not need numerical data, but semantic content that can be interpreted by the rules as we interpret them, because we want the story generation to be as close as possible to what humans might have done faced with similar sets of events.

We changed the meaning of the actions of the agents, not only by changing their names and the sets of them, as explained below in 3.3, but also by changing our interpretation of them, creating in this way a rather different world. For example, a value of "low" in economy has a particular meaning in the social simulation (a small house, no car), but in a Middle-Age time setting, a "low economy" means that the character is a peasant. Following this, the semantics we assign

to each fact affect the significance of that fact in particular. A "low economy" character in the medieval setting does not have the same interest than a "low economy" character in a modern society.

3.3 Adapting the MAS to a New Environment

Several minor changes have been introduced in the designed MAS for its adaptation to a new environment: a Fantasy Medieval World far from the previous Post-Modern context. Thus, we have introduced Name and Last Name apart from the ID of each agent, together with the inheritance of the Last Name: this will be useful for telling the stories of lineages, and for personal events. We added a new attribute to each individual: the race, so they can be elves, humans, dwarfs... Thanks to the modular structure of the system it has not been a difficult task to achieve.

Other changes are related to the system structure. One problem was the involvement of non natural deaths, never considered in the old MAS. We added a random possibility of dying for each agent, allowing the possibility that we can relate this early death to the betrayal of a friend, poisoning by a wife, or even a mysterious accident.

The finishing touches arrived with the recording of the sequence of "life events" for every individual. But usual life events, like having friends, finding a couple, or the birth of children, are not interesting enough to build an exciting fantasy adventure. Because of that, we have included new types of events related to this context that will appear randomly. Thus, along his path, the agent can suffer several spells (loss of memory, fireball... or even change of sex!), kill horrible monsters (ogres, dragons), get lost in mazes or dark forests, find treasures and magic objects in dangerous dungeons... In this way we can build a really amazing (and sometimes weird) story, with several characters that evolve and interact among them.

At the end of simulation, this collection of events, together with the agents' characteristics, is exported to an XML file. The XML-Schema pattern that rests beneath is not context-dependant, so the same format can be applied to other simulation environments. This file will be imported by a tool that will continue with the process of generating a story from the lives of some of these agents: the most interesting ones.

Here we present an explained example of the generated XML with the important information of each agent. In Figure 3 we can see the header of the file.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<Story Id="fantasy">
 <Description>
   A fantasy Middle-Age world
 </Description>
```

Figure 3. XML header of a log

Now the logs of every agent are listed: the initial ones, together with the next generations that appear during the simulation. Here we show just one of these logs: the one corresponding to the individual that will be selected as star of our story.

Each log is divided in two main sections. The first one (Figure 4) corresponds to the characteristics of the agent: each attribute of the character has two parameters, expressed as XML attributes: its ID (identifier of the attribute) and its Value. The value of these keys is, of course, context-dependant: here they represent aspects like its race or how religious the character is.

```
<Log Id="i212">
  <Description>
    Log of a character of the simulation.
  </Description>
  <Attribute Id="name"
      Value="badash"/>
  <Attribute Id="last_name"
      Value="taltaur"/>
  <Attribute Id="race"
      Value="elf"/>
  <Attribute Id="sex"
      Value="female"/>
  ...
  <Attribute Id="religion"
      Value="very religious"/>
```

Figure 4. Attributes of a character

The second main section (Figure 5) is the collection of life events, associated with the time in which they took place. As in the previous sections, XML attributes are context-free, but values of these attributes depend on the context. Thus, we can read in the full log (here we only show a small fragment), that in the year 515, the elf Badash Taltaur suffered a spell that transformed her into a frog. Or, analyzing the chain of events, we can see that the impossible love of her youth was, after she grew to be an adult, her formal couple, giving her many children and living happily... at least for some years.

```
  <Events>
   ...
   <Event Id="e9" Time="515"
      Action="spelled" Param="frog"/>
   <Event Id="e10" Time="515"
      Action="impossible love"
      Param="i229"/>
   ...
   <Event Id="e14" Time="526"
      Action="couple" Param="i229"/>
   <Event Id="e15" Time="526"
      Action="child" Param="i258"/>
   <Event Id="e16" Time="526"
  </Events>
</Log>
```

Figure 5. Events of a character

3.4 A Rule-Based Story Generation System

Given appropriate configuration parameters the social simulation generates a set of results which is sufficiently complex to constitute an interesting challenge for content planning. For this purpose, we have extended the manual story generation tool with an automatic story generation system. This program accepts a thread of facts from each agent of a defined set, and analyzes the connections and relations between these threads in time.

In our current design, we have chosen to perform an iteration through the elements of the log, using a rule-based system. Our first try was to implement the rule system in Jess [3], but, although it did work, the execution was extremely slow, and it required enormous amounts of memory. In contrast, writing rules in Jess is much easier than in Java, language in which we have developed the final version.

We have defined a set of domain-dependent rules for this problem in particular. We want to keep separated the general application of story generation (the editor *Herodotus*, structures for storing the stories, the natural language text generator, etc.) from ad-hoc content specialized for the specific system or a particular game. In this way the only work needed for adapting the application to other domains is restricted to defining the rules that establish which facts are important, and how they are going to appear in the final text, presentation or animation.

We have considered these rules to be expert knowledge. In the domain we are working on we cannot ignore the semantics present in the data saved in the logs during the gameplay for story generation. The meaning of particular attributes, not measured with numerical weights, must be taken into account before narrating a log: killing a red dragon is usually more interesting than killing a little spider. Of course, we can set some numerical values, as "kill-dragon interest", that should be a higher value than "kill-spider interest", but the final discourse will be made interesting with some "hand-made" rules, established by the system administrator, or perhaps the game-master.

3.4.1 Content Determination

As commented before, the first thing to do is to determine which data is not going to be told, and remove it. There are many possible solutions for this problem. The one we have used is to give a *factor of interest* to the characters. This interest factor is only a numerical value that represents how important it is for that character to appear in the story, not necessarily the comparative importance of that character with respect to other characters in the story. The value can represent real interest, coherence, fun, or any other reason why a given element from the logs should appear in the final text. In this way, a unimportant character can have a high *factor of interest*, because it is necessary that such character appears in the story. This factor is divided in two values:

- *Base interest* ($I_b(X)$) is the value we associate with the facts of some character X, and with their attributes. In this way, the character can be easily evaluated. With the attributes we can design a heuristic function h that represents the significance of some fact in the life of that character, given the attributes. It is usual for a man to fall in love, but not for an orc. That is why falling in love is more interesting in an orc's life than in a human's life. The actual method for computing $I_b(X)$ is shown in Formula 1 below,

$$I_b(X) = \sum_{i=1}^{n} f_i \cdot h\,(X, i) \tag{1}$$

  where f_i is the interest that we assign by hand for the fact i, x is the character, and $h(x, i)$ is the weight for the appearance of i in the life of x. The value of h is calculated with the type of i (what kind of fact it is) and with the attributes of x (if it is an elf, or an orc).
- *Relationship interest* ($I_r(X)$) is the level of importance of a character X calculated from the interest of their relationships with other characters: friends, foes, offspring, etc. We could not build a good factor of interest by considering only the characters as individuals, so we added this additional value. As before, the attributes of a character determine the final interest. We have a new function, g, depending on the relationship and the two characters, that represents the true interest of a relation: two elfs can easily be friends, but it is very strange (and perhaps we should tell about it) a friendship between an elf and an orc. The actual value is obtained using Formula 2 below,

$$I_r(X) = \sum_{i=1}^{n} I_b(Y) \cdot g\,(X, Y, i) \tag{2}$$

  where $I_b(Y)$ is the *base interest* of the character who has the relation i with X, and g is the heuristic function of the relationship i between different characters X and Y. The value of g is calculated with the type of i (what kind of fact it is) and with the attributes of x and y (if they are two orcs, or an orc and an elf, for example).

The final *factor of interest* is, in our current implementation, obtained according to Formula 3:

$$I_f = I_b + I_r \tag{3}$$

Once we have this value calculated, we have a new explicit data that will determine what is going to appear in the final structure. With the "interest" and some rules, like redundancy elimination (delete symmetric data: A is friend of B, and B is friend of A, then delete A or B), omission of irrelevant characters (those that are just born at some stage of the gameplay and then die at some later stage with little intervening activity), and of course, an importance filter (remove those characters whose *factor of interest* falls below a given threshold), we can have a set of facts and characters ready to form part of the final story. With these and other rules and filters, we can determine not only which characters are going to appear, but also which of their facts are going to be shown. The particular solution applied in generating the interest factor ensures that facts that are related to important characters are always included. This is intended to avoid the risk of eliminating non-interesting elements that may be of importance in a plot.

3.4.2 Discourse Planning

In discourse planning, basically we just reorder the facts in the story, and adapt the relationships between them. This is, in terms of computing, an easy task. But the goal of discourse planning is not only organizing the facts stored in the log, but inferring the guidelines of the story, giving them priority, and making them the main structure of the narration.

Several tasks must be accomplished in order to create a meaningful, clear and interesting story. In fact, we have found that these tasks are very dependent on the domain, and on what we want to present in the final story. While, as we have verified, adjusting the *factor of interest* to appropriate values is usually good enough for *content determination*, in discourse planning this is not true. It is very difficult to write general rules that generate different stories for different domains.

What we have done is to define ad-hoc rules for the domain we are working on, to process the particular data we have; and rules to generate the stories that we think that could be interesting for the reader. This rules are based on the three sets of data that we have: *facts content*, *attributes of the characters* and the *time*.

Some of this rules are, for example, to narrate the birth and death date of the main character only, to maintain a more or less time-ordered discourse, to talk about the unusual facts only, and so on. If we wanted to generate stories of fairy tales, for example, we could have omitted the dates, and we could have ordered the facts in a different way, trying to hide data that is only important in the end of the story.

It is important the way we manage time. In [1] we can see many ways of representing time, very related to this work. At this moment we consider that facts are instantaneous, ignoring intervals and time reasoning. We generate the time nexus between facts also with rules, and we have verified that, for simple narrations, this could be sufficient.

Once we processed the initial log, and having performed *content determination* and *discourse planning*, we can generate the final result. This result can be not only text, but also a script that controls an animation, a generated comic, or a summarised reproduction of the gameplay.

3.4.3 Sentence Planning

The final generation of the story is not only a nice way of showing the results. It can make the discourse interesting or boring, even if the order of the facts resulting from *discourse planning* is bad or good, respectively. Thus, we cannot ignore this step if we want to evaluate the generated content. It is not the same to say "Elrond was an Elf. He had a daughter called Arwen. Elrond was friend of Aragorn.", as to say "Elrond the Elf, father of Arwen, was friend of Aragorn the King". The final form of sentences not only gives beauty to the text, but may also convey information not actually present in the data structure. We can infer, in the second sentence, that Elrond is somebody important, as Arwen, and Aragorn is going to play a main role in the story. This knowledge is not contained in the first sentence. To achieve computational modeling of these characteristics is currently beyond the scope of this paper, but we intend to address it in future work.

The actual examples of output text presented in this paper have been generated with the use of a simple template-based surface realizer built on purpose for this particular application, and which produces monotonous text with little inflexion and no concern for literary style. This is because the main concern of the research reported here has been the succesful completion of the content determination and discourse planning tasks. For this purpose, such output texts are sufficient, and yet considerably easier for the reader to understand than the corresponding XML output files. The final result in terms of stories to be read by humans may be considerably improved by resorting to an existing sentence planning application. In future work, we intend to address this problem by integrating the present work with the PRINCE generator [8].

3.5 An Example

Now, we show a real example of our application. The multi-agent system is capable of running parametrized simulations, changing the number of characters, probabilities of the facts, years of simulation, and all other attributes of the system. Once executed, the system generates logs in XML, like the ones we have presented in 3.3.

At this stage, the story generation application reads the resulting XML file, and outputs a text. This example is the result of a simulation of the life of 200 initial characters and their descendants over a time span of 80 years. The system has inferred who is the most important character, and it produces the following rendition of her mortal life:

The Great Story - A fantasy Middle-Age world:
Badash Taltaur the Elf was born in 504.
Badash Taltaur met Amdor Taltaur, and she was lost in a forest, then she was enchanted with the incredible spell of memory, then she found a Magic Ring.
Badash Taltaur was lost in a labyrinth, then she met Werlom Mcknight, and Werlom Mcknight was offspring of Rirbag Greatgibber, and Badash Taltaur was involved in a great battle, then she was enchanted with the incredible spell of frog.
Badash Taltaur fell in love, desesperately, with Werlom Mcknight, then she was lost in a forest, then she found a Treasure, then she married Werlom Mcknight, then she had a child: Idrin Taltaur.
Badash Taltaur had a child: Dora Taltaur, then she had a child: Dwalin Taltaur, then she had a child: Pimmam Taltaur, then she had a child: Baradadan Taltaur, then she found a Magic Sword.
Badash Taltaur found a Magic Ring, then she was lost in a forest, then she was involved in a great battle, then she was enchanted with the incredible spell of sex, then she was lost in a forest.
Badash Taltaur found a Treasure.
Badash Taltaur died in a mysterious accident in 555.
The end.

4 Discussion

There are three main points worth discussing in an analysis of the proposed story generation solution: the possibility of evaluating results by comparing with human performance over similar tasks, the possible role of the sentence planning solution employed in the perceived quality of the output, and the particular choice of implementation that has been used.

4.1 Evaluation Against Human Performance

We are not evaluating if the story is interesting or funny, yet. We are only focusing on how similar are the machine generated stories with those stories that could be written by humans from the same source. We will keep on refining, in particular, the *content determination* process, because the output of this step is where we decide the interest of the elements of the story.

It would be interesting to compare the resulting work of the application of *content determination* and *discourse planning* in a log from a gameplay presented on this paper with a manual generation of the same log. In this way, we could see if the rules that we have applied in the code (filtering, ordering, connections between events) are those which would be applied by a human narrator. This task is, of course, possible, but the cost in time and human effort is very high. To perform the previous tasks by hand, over a log of 500 characters, could mean several days of work.

This prevents us, in principle, from evaluating how correct our application is, but it is an indicator of the utility of this work. This kind of story generation is very hard to do by humans, and can be easily done by machines. However, one possible evaluation of the system could be to ask a group a people to write a text describing a small set of facts of the log. This would provide an evaluation of the discourse planning stage of the system, but only partially address the evaluation of content determination - unless an evaluator chooses to omit a fact included in the selected set. In this way, we could compare human generated texts with machine generated ones.

4.2 The Effect of Bad Sentence Planning on Perceived Quality

Relative to the final output of the present work, it is obvious that the final example of generated text that we have presented does not have

a nice form, and the narration is a little boring. The reason is that the *sentence planner* we are using is a skeleton implementation not even intended to be passably correct at its task.

This can be easily illustrated by a close analysis of the sentence planning tasks that are performed poorly in the given example, and considering how the text might have improved if those tasks were actually addressed in the implementation.

An important issue is how the sentence planner decides to represent the fact that a particular set of facts have been grouped by the discourse planner into a block of related events, to be narrated as a distinct thread within the discourse. In the current implementation this is simply solved by chunking all such facts into a single sentence, clumsily linked together with discourse markers indicating some kind of sequence. This can be seen in the example above in fragments such as:

Badash Taltaur met Amdor Taltaur, and she was lost in a forest, then she was enchanted with the incredible spell of memory, then she found a Magic Ring.

This could easily be improved if, for instance, a simple sequence of sentences where used:

Badash Taltaur met Amdor Taltaur. She was lost in a forest. She was enchanted with the incredible spell of memory. She found a Magic Ring.

However this obscures the fact that there are indeed chronological relations linking these particular facts with one another. A complex sentence planner would have to take this into account, and possible decide to give up the chronological information in favour of more fluid text.

Another related problem concerns sentence aggregation. The current sentence planner is incapable of detecting that a fragment such as:

...then she married Werlom Mcknight, then she had a child: Idrin Taltaur.
Badash Taltaur had a child: Dora Taltaur, then she had a child: Dwalin Taltaur, then she had a child: Pimmam Taltaur, then she had a child: Baradadan Taltaur,

might be considerably easier to read in a form like:

She married Werlom McKnight. They had five children: Idrin Taltaur, Dora Taltaur, Dwalin Taltaur, Pimmam Taltaur and Baradadan Taltaur.

This transformation seems simple but involves at least an abstraction that is not trivial: the fact that a set of facts with the same predicate can be regrouped as a single predicate with a plural compound second argument.

This same example illustrates a different problem, that of referring expression generation. The sentence planner does indeed address this task in a clumsy manner, deciding at different places in the discourse to refer to a given character either by its full name or by a pronoun. This could be greatly improved, especially if it were considered in its interaction with elements such as additional sentence boundaries arising from a more refined realization of narrative threads. Additional issues related with this task arise from the fact that, if they are mentioned in close proximity, knowing the surname of the parents one may omit the surnames of all their children. This could lead to an even more refined version of the example above:

She married Werlom McKnight. They had five children: Idrin, Dora, Dwalin, Pimmam and Baradadan.

4.3 Implementation Issues: Modularity vs. Efficiency

Relative to the implementation, it is also worth discussing the efficiency problems we have encountered using a declarative rule definition system like Jess. We first tried to build the whole rule system, and the evaluation of every fact present in the log, just using an implementation written in Jess. But it has problems of efficiency, because the algorithm behind Jess (the *Rete* algorithm), works in a way that is not optimal for our problem in particular.

We could have, then, implemented a hybrid system, and, while this is possible, the remaining content that could have been written in Jess was very reduced and easily translatable to Java. For that reason, we decided to stop using Jess, at least for this work.

As an example of rule, we present a definition of a simple filter that removes from the list of facts, those whose interest is equal to zero.

In Figure 6 we show the code as we implemented using Jess. The line "`(event (type ?type)(interest 0))`" means "that event of a defined type that has no interest". The other conditions in the rule are needed for the interface with Java (with the data structures). The resulting action of the rule is to remove, from the story, that fact.

```
(defrule remove-non-interesting
    (story (OBJECT ?story) (facts ?facts))
    (fact (type ?type)(OBJECT ?fact))
    (test (?facts contains ?fact))
    (event (type ?type)(interest 0))
    =>
    (?story remove ?fact)
    )
```

Figure 6. Rule implemented in *Jess*

The corresponding code in Java is the one we show in Figure 7. This implementation is much faster. If we add more rules to the system, and make them sequential in a Java program, it will be even more efficient than if we implement the rules in Jess.

```
ListIterator<Fact> it = facts.listIterator();
while (it.hasNext()) {
  Fact h = it.next();
  if (h.getInterest() == 0) {
    it.remove();
  }
}
```

Figure 7. Rule implemented in *Java*

5 Conclusions

We have presented a system where interactions between agents over a long period of time can be told in natural language automatically. With this work MMORPGs can generate texts describing the gameplay for different audiences and purposes. The text could be generated at the end of the game or while a player is still playing, or it could be the script for a 3D, or a generated comic.

We have shown a particular way of generating the stories, based on rules. We have explained a three-step process for performing this task, and we have verified that for *discourse planning*, the rule-system is very dependent on the domain, and the desired type of story.

Although the implementation includes an application for the manual development of narrative structures from a log of events, it has proved impossible to contrast the results generated by the application with any manually obtained equivalent due to the sheer size of the input logs that the application is currently handling. The effort involved for human evaluators is too large for voluntary participation.

The results of the system are less impressive - when rendered in a readable text format - than they might have been if the system included an elaborate sentence planning module. The current version is just a skeleton implementation that lets down an otherwise acceptably selected and planned discourse.

6 Future work

We plan to empower the multi-agent system, through several lines of evolution. The main point where improving is always required is to build a more interesting story. The introduction of random events was a huge step in this direction, and more improvements in this field can have incredible results.

We can add more characteristics to the agents, selecting the most attractive for the context. For example, including the profession or role of each agent could be a great idea for improving the story told: knight, king, princess, wizard, priest, peasant... If a peasant kills a dragon, would be much more heroic than if a knight does so. Another good characteristic to be introduced is geographical position. In our social simulation there is a graphical visualization of the agents, distributed in a space. If we parse this (x, y) positions dividing the space into countries, we would have knights that come from a far kingdom to save the princess.

Adding characteristics is now a particular field of the agents... but what about if we give "personality" to the inanimate objects? If we give an ID and a Name to the objects of the events, we would have events like: "lost in the Lorien Forest", "found the Anduril sword", or "killed by the dragon Smaug". These events can be analyzed to generate stories in which the dragon Smaug killed three knights (with their names), but the fourth one, Aragorn, at last killed him and freed the Gondor kingdom.

The relationships between agents represent another sector where we can add complexity. New type of relations could be included: hate (natural feeling between orcs and elfs), complex family relationships (like cousins), to belong to the same religious order...

The most part of the fantastic life events (like killing a dragon) are generated randomly in every agent. Thus, the events are particular for each agent. A new type of event could be generated: a common random event, which could affect to lots of agents at the same time (maybe to the whole world, maybe to just one kingdom). For example, a huge battle in the year 527 between dwarfs and orcs, killing lots of them, harming others, killing loved ones... and even it can lead to a prince that inherit the crown of his dead father.

Other improvements are planned for the story generation tool. A new objective can be to find a more efficient alternative to the one we tried with Jess only, perhaps a hybrid implementation between the speed of a procedural language, and the flexibility and power of a rule definition language, so the tool can be built in a more modular way, and also having the benefit of an easier to write system. Of course, another line of evolution is to enlarge the amount of rules that control the rule-based system, so more precise and complex knowledge can be used.

Another important objective is to apply more sophisticated time representation and reasoning concepts for fact and block nexus. It is very important to focus on how we narrate the story in terms of choosing what should be told before, and how we connect it with the rest of the discourse.

Different approaches to story generation are planned, and future comparisons between this work and them. An interesting line of research that is contemplated is to consider whether a Case-Based Reasoning solution, applying in discourse planning a set of patterns learned from the way humans have told similar sequences of events in human-generated stories, might compete with the simple rule-based solution.

ACKNOWLEDGEMENTS

This work is partially supported by the Spanish Ministry of Education and Science project TIN2006-14433-C02-01, and research group grant UCM-CAM-910494, jointly funded by Universidad Complutense de Madrid and the Comunidad Autónoma de Madrid (Dirección General de Universidades e Investigación).

REFERENCES

[1] James F. Allen, 'Time and time again: the many ways to represent time', *International Journal of Intelligent Systems*, **6**, 341–355, (1991).
[2] M. E. Bratman, *Intention, Plans, and Practical Reason*, Harvard University Press, Cambridge, MA, 1987.
[3] Ernest Friedman-Hill. Jess, the rule engine for the java platform. http://herzberg.ca.sandia.gov/jess/, 2006.
[4] E. Hovy, 'Automated discourse generation using discourse structure relations', *Artificial Intelligence*, **63**(1-2), 341–386, (1993).
[5] W. Mann and S. Thompson, 'Rhetorical structure theory: Towards a functional theory of text organization', *Text*, **3**, 243–281, (1988).
[6] K. McKeown, 'Discourse strategies for generating natural language text', *Artificial Intelligence*, **27**, 1–42, (1985).
[7] J. Pavon, M. Arroyo, S. Hassan, and C. Sansores, 'Simulacion de sistemas sociales con agentes software', in *Actas del Campus Multidisciplinar en Percepcion e Inteligencia, CMPI-2006*, volume I, pp. 389–400, (2006).
[8] Francisco C. Pereira, Raquel Hervás, Pablo Gervás, and Amilcar Cardoso, 'A multiagent text generator with simple rhetorical habilities', in *Proc. of the AAAI-06 Workshop on Computational Aesthetics: AI Approaches to Beauty and Happiness, July 2006*. AAAI Press, (2006).
[9] E. Reiter and R. Dale, *Building Natural Language Generation Systems*, Cambridge University Press, 2000.